

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

IN RE APPLICATION OF: Kensaku YAMAGUCHI, et al.

GAU:

SERIAL NO: NEW APPLICATION

EXAMINER:

FILED: HEREWITH

FOR: METHOD FOR USING SHARED LIBRARY IN TAMPER RESISTANT MICROPROCESSOR

REQUEST FOR PRIORITY

COMMISSIONER FOR PATENTS
ALEXANDRIA, VIRGINIA 22313

SIR:

☐ Full benefit of the filing date of U.S. Application Serial Number _____, filed _____, is claimed pursuant to the provisions of **35 U.S.C. §120**.

☐ Full benefit of the filing date(s) of U.S. Provisional Application(s) is claimed pursuant to the provisions of **35 U.S.C. §119(e)**:
Application No. _____ **Date Filed** _____

☒ Applicants claim any right to priority from any earlier filed applications to which they may be entitled pursuant to the provisions of **35 U.S.C. §119**, as noted below.

In the matter of the above-identified application for patent, notice is hereby given that the applicants claim as priority:

COUNTRY

APPLICATION NUMBER

MONTH/DAY/YEAR

Japan

2002-192422

July 1, 2002

Certified copies of the corresponding Convention Application(s)

☒ are submitted herewith

☐ will be submitted prior to payment of the Final Fee

☐ were filed in prior application Serial No. _____ filed _____

☐ were submitted to the International Bureau in PCT Application Number _____
Receipt of the certified copies by the International Bureau in a timely manner under PCT Rule 17.1(a) has been acknowledged as evidenced by the attached PCT/IB/304.

☐ (A) Application Serial No.(s) were filed in prior application Serial No. _____ filed _____; and

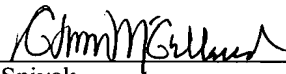
☐ (B) Application Serial No.(s) _____

☐ are submitted herewith

☐ will be submitted prior to payment of the Final Fee

Respectfully Submitted,

OBLON, SPIVAK, McCLELLAND,
MAIER & NEUSTADT, P.C.



Marvin J. Spivak

Registration No. 24,913



22850

Tel. (703) 413-3000
Fax. (703) 413-2220
(OSMMN 05/03)

C. Irvin McClelland
Registration Number 21,124

日 本 国 特 許 庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出 願 年 月 日
Date of Application:

2002年 7月 1日

出 願 番 号
Application Number:

特願2002-192422

[ST.10/C]:

[JP2002-192422]

出 願 人
Applicant(s):

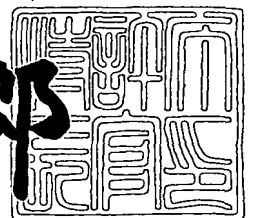
株式会社東芝



2003年 2月21日

特 許 庁 長 官
Commissioner,
Japan Patent Office

太田 信一郎



出証番号 出証特2003-3009672

【書類名】 特許願

【整理番号】 13B022121

【提出日】 平成14年 7月 1日

【あて先】 特許庁長官殿

【国際特許分類】 G09C 1/00
G06F 9/26
H03M 13/45

【発明の名称】 耐タンパプロセッサにおける共有ライブラリの使用方法
およびそのプログラム

【請求項の数】 14

【発明者】
【住所又は居所】 神奈川県川崎市幸区小向東芝町 1 株式会社東芝 研究
開発センター内
【氏名】 山口 健作

【発明者】
【住所又は居所】 神奈川県川崎市幸区小向東芝町 1 株式会社東芝 研究
開発センター内
【氏名】 橋本 幹生

【特許出願人】
【識別番号】 000003078
【氏名又は名称】 株式会社 東芝

【代理人】
【識別番号】 100083806
【弁理士】
【氏名又は名称】 三好 秀和
【電話番号】 03-3504-3075

【選任した代理人】
【識別番号】 100068342
【弁理士】

【氏名又は名称】 三好 保男

【選任した代理人】

【識別番号】 100100712

【弁理士】

【氏名又は名称】 岩▲崎▼ 幸邦

【選任した代理人】

【識別番号】 100100929

【弁理士】

【氏名又は名称】 川又 澄雄

【選任した代理人】

【識別番号】 100108707

【弁理士】

【氏名又は名称】 中村 友之

【選任した代理人】

【識別番号】 100095500

【弁理士】

【氏名又は名称】 伊藤 正和

【選任した代理人】

【識別番号】 100101247

【弁理士】

【氏名又は名称】 高橋 俊一

【選任した代理人】

【識別番号】 100098327

【弁理士】

【氏名又は名称】 高松 俊雄

【手数料の表示】

【予納台帳番号】 001982

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 耐タンパプロセッサにおける共有ライブラリの使用法およびそのプログラム

【特許請求の範囲】

【請求項 1】 暗号化されたコードを復号化して実行する機能を有すると共に、少なくとも 1 つのプログラム及びそのプログラムが呼び出す少なくとも 1 つの共有ライブラリに対応した複数の暗号鍵を格納する複数の領域からなるテーブルを有するマイクロプロセッサ上で、呼出元プログラムから前記少なくとも 1 つの共有ライブラリを呼び出して使用する方法であって、

共有ライブラリ用のタスクを作成し、

このタスクにタスク識別子を割り当て、

共有ライブラリのヘッダから命令鍵を取得し、

この命令鍵を前記マイクロプロセッサ内の前記共有ライブラリに割り当てられた識別子に対応するテーブルの領域に格納し、

共有ライブラリ内のローダーを実行して初期化を行ない、

共有ライブラリ内のエントリポイントを経由して呼出元へ制御を返すことからなる共有ライブラリの使用法。

【請求項 2】 請求項 1 記載の共有ライブラリの使用法であって、前記命令鍵をマイクロプロセッサ内のテーブルの領域に格納した後で、共有ライブラリ内のインポートテーブルを参照して他の共有ライブラリをロードすることを特徴とする共有ライブラリの使用法。

【請求項 3】 暗号化されたコードを復号化して実行する機能を有すると共に、少なくとも 1 つのプログラム及びそのプログラムが呼び出す少なくとも 1 つの共有ライブラリに対応した複数の暗号鍵を格納する複数の領域からなるテーブルを有するマイクロプロセッサ上で、呼出元プログラムから前記少なくとも 1 つの共有ライブラリを呼び出して使用する方法であって、

共有ライブラリ用のタスクを作成し、

このタスクにタスク識別子を割り当て、

共有ライブラリのヘッダから命令鍵を取得し、

この命令鍵を前記マイクロプロセッサ内の前記共有ライブラリに割り当てられた識別子に対応するテーブルの領域に格納し、

共有ライブラリ内のローダーを呼出元の個数だけ実行して初期化を行ない、

共有ライブラリが使用するデータを暗号化するためのデータ鍵を呼出元の個数だけ作成し、

このデータ鍵を上記プロセッサ内の当該共有ライブラリに対応する識別子が割り当てられた前記テーブルの領域に格納し、

共有ライブラリ内のエントリポイントを経由して呼出元へ制御を返し呼出元からの呼出を待機し、

呼出元と認証をし、

呼出元が作成した共有メモリ領域のアドレスを受け取り、

上記共有メモリ領域を呼出元とのデータ交換に使用する共有暗号化データ領域に設定し、

呼出元から共有ライブラリ内のサブルーチン呼出信号を受け取り、

呼出元のチェックサムを検証し、

呼出元とのチェックサムが共有ライブラリのチェックサムと一致した場合に要求された処理を行ない、

上記処理の結果を前記共有暗号化データ領域にチェックサムを加えて送付することからなる共有ライブラリ使用方法。

【請求項 4】 請求項 3 記載の共有ライブラリ使用方法であって、

呼出元のチェックサムの検証を行った後で、呼出元から要求された処理を行うための作業メモリ領域を共有ライブラリのデータ鍵で暗号化し、

この暗号化された作業メモリ領域内で呼出元から要求された処理を行うことを特徴とする共有ライブラリ使用方法。

【請求項 5】 請求項 3 記載の共有ライブラリ使用方法であって、

前記呼出元との認証は、共有ライブラリに予め格納されている秘密鍵による署名を付加した認証情報を呼出元へ送付し、呼出元との間で使用する共通鍵を作成することで行われることを特徴とする共有ライブラリ使用方法。

【請求項 6】 請求項 5 記載の共有ライブラリ使用方法であって、

前記呼出元と認証を行った後で前記共通鍵を共有ライブラリ内の前記データ鍵で暗号化し、

このデータ鍵で暗号化された共通鍵を共有ライブラリの秘匿データ領域に格納することを特徴とする共有ライブラリの使用方法。

【請求項 7】 暗号化されたコードを復号化して実行する機能を有すると共に、少なくとも 1 つのプログラム及びそのプログラムが呼び出す少なくとも 1 つの共有ライブラリに対応した複数の暗号鍵を格納する複数の領域からなるテーブルを有するマイクロプロセッサ上で、呼出元プログラムから前記少なくとも 1 つの共有ライブラリを呼び出して使用する方法であって、

共有ライブラリ用のタスクを作成し、

このタスクにタスク識別子を割り当て、

共有ライブラリのヘッダから命令鍵を取得し、

この命令鍵を前記マイクロプロセッサ内の前記共有ライブラリに割り当てられた識別子に対応するテーブルの領域に格納し、

共有ライブラリ内のローダーを実行して初期化を行ない、

前記マイクロプロセッサ内の乱数生成部から取得した乱数から当該共有ライブラリの作業用メモリ領域を暗号化する為のデータ鍵を生成し、

共有ライブラリ内のエントリポイントを経由して呼出元へ制御を返し呼出元からの呼出を待機することからなる共有ライブラリの使用方法。

【請求項 8】 暗号化されたコードを復号化して実行する機能を有すると共に、少なくとも 1 つのプログラム及びそのプログラムが呼び出す少なくとも 1 つの共有ライブラリに対応した複数の暗号鍵を格納する複数の領域からなるテーブルを有するマイクロプロセッサ上で、

共有ライブラリ用タスクを作成するステップと、

このタスクにタスク識別子を割り当てるステップと、

共有ライブラリのヘッダから命令鍵を取得するステップと、

この命令鍵を前記マイクロプロセッサ内の前記共有ライブラリに割り当てられた識別子に対応するテーブルの領域に格納するステップと、

ローダーを実行して初期化を行うステップと、

エントリポイントを経由して呼出元へ制御を返し待機するステップを実行させるためのプログラム。

【請求項 9】 暗号化されたコードを復号化して実行する機能を有すると共に、少なくとも 1 つのプログラム及びそのプログラムが呼び出す少なくとも 1 つの共有ライブラリに対応した複数の暗号鍵を格納する複数の領域からなるテーブルを有するマイクロプロセッサ上で、

共有ライブラリ用のタスクを作成するステップと、

このタスクにタスク識別子を割り当てるステップと、

共有ライブラリのヘッダから予め記録されている命令鍵を取得するステップと

この命令鍵を前記マイクロプロセッサ内の前記共有ライブラリに割り当てられた識別子に対応するテーブルの領域に格納するステップと、

共有ライブラリ内のローダーを実行して初期化を行うステップと、

上記ローダーで初期化をする際にこの共有ライブラリが使用するデータを暗号化するデータ鍵を作成するステップと、

このデータ鍵を上記プロセッサ内の当該共有ライブラリに対応する識別子が割り当てられた前記テーブルの領域に格納し、

共有ライブラリ内のエントリポイントを経由してアプリケーションへ制御を返すステップと、

呼出元と認証を行うステップと、

呼出元が作成した共有メモリ領域のアドレスを受け取るステップと、

上記共有メモリ領域を呼出元とのデータ交換に使用する共有暗号化データ領域に設定するステップと、

呼出元から共有ライブラリ内のサブルーチン呼出信号を受け取るステップと、

呼出元のチェックサムを検証するステップと、

呼出元とのチェックサムが共有ライブラリのチェックサムと一致した場合に要求された処理を行うステップと、

上記処理の結果を前記共有暗号化データ領域にチェックサムを加えて送付するステップとを実行させるためのプログラム。

【請求項 1 0】 暗号化されたプログラムを該プログラム単位で実行する機能を有するマイクロプロセッサ上で実行される呼出元プログラムが、共有ライブラリを利用する方法であって、

前記呼出元プログラムは固有の第 1 の暗号鍵で暗号化されており、

前記共有ライブラリは固有の第 2 の暗号鍵で暗号化されており、

前記共有ライブラリ該共有ライブラリの中に該共有ライブラリ自身の秘密鍵を保持し、

前記呼出元プログラムは前記秘密鍵に対応する公開鍵を保持し、

前記呼出元プログラムは第 1 の一時鍵を該呼出元プログラム内部で生成し、この第 1 の一時鍵に基づいて生成した第 1 の鍵共有情報を前記共有ライブラリに通知し、

前記共有ライブラリは第 2 の一時鍵を該呼出元プログラム内部で生成し、この第 2 の一時鍵に基づいて生成した第 2 の鍵共有情報および受け取った前記第 1 の鍵共有情報に対して前記秘密鍵に基づいて署名を生成し、前記第 2 の認証用情報および前記署名を前記呼出元プログラムに通知し、

前記共有ライブラリは前記第 2 の一時鍵および受信した第 1 の鍵共有情報から共有鍵を生成し、

前記呼出元プログラムは前記第 1 の一時鍵および受信した第 2 の鍵共有情報から共有鍵を生成し、

前記呼出元プログラムは受け取った前記署名を、前記第 1 の鍵共有情報及び第 2 の鍵共有情報について前記公開鍵を用いて検証し、前記検証に成功したときのみ、前記共有鍵を用いて暗号化した処理対象のデータを共有ライブラリに通知し、前記共有ライブラリは、前記通知された処理対象のデータを共有鍵により復号化して所定の処理を行い、この処理結果として得られたデータを、前記共有鍵を用いて暗号化して前記呼出元プログラムへ通知することを特徴とする共有ライブラリの使用方法。

【請求項 1 1】 請求項 1 0 記載の共有ライブラリの使用方法であって、

前記マイクロプロセッサには暗号化されたデータの暗号化鍵及びメモリ領域を指定する特殊レジスタが設けられ、

前記送信元プログラムは、前記データの交換に先立って第 1 のデータ交換用メモリ領域を前記特殊レジスタに設定し、

前記共有ライブラリは、前記データの交換に先立って第 2 のデータ交換用メモリ領域を前記特殊レジスタに設定し、

前記送信元プログラム及び前記共有ライブラリは前記データの送付及び受信の際に前記第 1 及び第 2 のデータ交換用メモリ領域への書き込みを行うことにより暗号化された前記データを前記メモリ領域に書き込むことを特徴とする共有ライブラリ使用方法。

【請求項 1 2】 請求項 1 1 記載の共有ライブラリ使用方法であって、

前記共有ライブラリは前記第 1 の一時鍵の生成時に所定の乱数生成手段により作業鍵を作成し、

前記作業鍵及び作業用メモリ領域を前記特殊レジスタに設定し、

前記共有ライブラリ内部における一時データの保持には前記作業用メモリ領域を使用することにより、一時データを暗号化して前記作業用メモリ領域で指定されたメモリ領域に格納することを特徴とする共有ライブラリ使用方法。

【請求項 1 3】 暗号化されたプログラムを該プログラム単位で復号化して実行する機能を有するマイクロプロセッサ上で実行される呼出元プログラムが、共有ライブラリを利用する方法であって、

前記マイクロプロセッサは該マイクロプロセッサ毎に固有のプロセッサ秘密鍵を持ち、

前記マイクロプロセッサは複数の前記プログラムに各々対応する複数の配布鍵を、前記マイクロプロセッサ秘密鍵によってそれぞれ復号化して、複数のプログラム鍵を取り出し、

前記各々のプログラム鍵により暗号化された状態で外部メモリに配置された複数のプログラムを、対応する前記プログラム鍵によってそれぞれ復号化して疑似並列的に実行し、

前記マイクロプロセッサ内部に疑似並列実行単位の制御情報を保持するテーブルを持ち、このテーブルの識別子に基づいて、前記マイクロプロセッサ内部に保持する情報のアクセス制御を行い、

前記呼出元プログラムから呼び出される共有ライブラリの読み込み時に前記実行制御情報を保持するエントリを割り当て、他のプログラムとは個別に実行制御することを特徴とする共有ライブラリ使用方法。

【請求項 1 4】 請求項 1 3 記載の共有ライブラリ使用方法であって、

前記制御情報はコンテキスト情報を暗号化するための暗号鍵であることを特徴とする共有ライブラリ使用方法。

【発明の詳細な説明】

【 0 0 0 1 】

【発明の属する技術分野】

本発明は、マルチタスクのプログラム実行環境を支援する機能を持つマイクロプロセッサで、プログラムやデータの暗号化及び復号化機能を有し、プログラムの実行コードや処理対象であるデータの秘匿性の保護、及び改竄の防止を可能とするマイクロプロセッサにおける、共有ライブラリの使用方法およびそのプログラムに関する。

【 0 0 0 2 】

【従来の技術】

近年のコンピュータシステムでは、パーソナルコンピュータをはじめとして、様々なメーカーのハードウェアやソフトウェアを組み合わせで構築される、オープンシステムが広く普及している。オープンシステムでは、ハードウェアやシステムプログラム（又はオペレーティングシステム、以下OSと呼ぶ）の情報が開示されており、エンドユーザがこの情報をもとにシステムプログラムを改変・改竄することも原理的に可能である。

【 0 0 0 3 】

このような環境で実行されるアプリケーションプログラムにおいて、その提供者が、プログラムを解析や改竄から完全に保護することは困難である。アプリケーションプログラムはOSの管理下で動作するため、このOS自体が改変されて攻撃の手段として使われた場合、これを避ける手段がない。

【 0 0 0 4 】

そこで予め、オープンシステムで動作するアプリケーションプログラムの解析

を防止するために、これを暗号化するという方法がある。プログラムを暗号化すると、解析が困難になるだけでなく、そのプログラムを改変した場合の動作が予測困難となるため、改竄の防止にも有効である。ただし暗号化したアプリケーションプログラムを既存のコンピュータで実行することはできないので、これを復号化しつつ実行するようなマイクロプロセッサが必要となる。このマイクロプロセッサは、OSがアプリケーションプログラムに対して敵対動作をとることを前提として、プログラムの秘密を守るものでなければならない。

【0005】

このような要件を満たすマイクロプロセッサとしては、橋本他による特開2001-230770や、Lie他による"Architectural Support for Copy and Tamper Resistant Software" (Computer Architecture News 28(5) p. 168-) がある。これらで提案されているマイクロプロセッサは、プログラムだけでなくプログラムが扱う情報やデータも暗号化して、解析や改変から保護する機能を備えている。また、保護を受けたプログラムを、複数同時に疑似並列実行するマルチタスクのプログラム実行環境を提供する。以下、このようなマイクロプロセッサを耐タンパマイクロプロセッサと呼ぶ。

【0006】

従来提案されてきた耐タンパマイクロプロセッサでは、アプリケーションプログラムは単体で動作し、その実行コードだけで必要な全ての処理が可能であるものとみなされてきた。また、複数のプログラムが協調動作をするためデータ領域の共有方法は寺本他による特願2000-402672で提案されている。しかし、この場合でも協調動作をしている各プログラムは、それぞれ独立した個別のプログラムである。

【0007】

一方、現代のOSでは共有ライブラリが使われることが多い。ライブラリとは、プログラムを構成する部分であるサブルーチン（プログラムにおいてなんらかの機能を有する一纏まりの命令群）等のサブプログラムを集めたものである。アプリケーションプログラムのプログラマは、アプリケーションプログラムの動作に必要な全ての機能をアプリケーションプログラムに実装するかわりに、一部の機

能をライブラリに用意されたサブプログラムに依存することができる。ライブラリとアプリケーションプログラムは、それぞれ別々に開発してから後で自由に組み合わせて使用することができるため、開発を効率化することができる。

【0008】

古典的なライブラリは、アプリケーションプログラムの作成時にアプリケーションプログラムにリンクされ、ライブラリのサブプログラムはアプリケーションプログラムと一体となって配布される。一方、今日広く使われているライブラリは、アプリケーションプログラムとは別ファイルとして配布される。

【0009】

共有ライブラリの場合、アプリケーションプログラムとのリンクは、ユーザがプログラムを実行する時に初めて行なわれる。またこのリンク動作は、アプリケーションプログラムの実行可能オブジェクトファイルに対してではなく、アプリケーションプログラムのメモリ上のイメージに対して行なわれる。いったんアプリケーションプログラムと共有ライブラリ間のリンクが行なわれると、共有ライブラリのサブプログラムは通常の共有ライブラリのサブプログラムと同様、アプリケーションプログラム本体から自由に呼び出して使用することができる。

【0010】

共有ライブラリを使う利点として、必要な記憶領域を節約できることが挙げられる。1つのアプリケーションプログラムとそれが使う共有ライブラリのサイズを合計すれば、共有ライブラリを使わない場合に比べてサイズが小さくなることはない。しかし、同じ共有ライブラリを使用するアプリケーションプログラムが複数ある場合は、共有ライブラリのコピーは1つだけあれば良いので、全体として記憶領域の節約になる。この記憶領域の節約は、アプリケーションプログラムの配布ファイルが置かれる二次記憶（ディスク等の外部記憶装置）にも、アプリケーションプログラムが実行時に置かれるコンピュータのメインメモリにも適用できる。

【0011】

共有ライブラリの中でも動的リンク型と呼ばれる種類のもの（動的リンク共有ライブラリ）は、アプリケーションプログラムを変更することなく共有ライブラ

りを変更できることを特徴としている。動的リンク共有ライブラリを使うと、ある共有ライブラリを別のものと置き換えることにより、アプリケーションプログラム本体を変更することなくその機能の一部分を変更したり、不具合を修正したりすることが可能である。また、アプリケーションプログラムが実行開始後に使用可能な共有ライブラリを検索し、これをロードするようにすれば、共有ライブラリのみを別途提供することにより、アプリケーションプログラム本体に変更を加えることなく機能追加が可能になる。このような使用形態の共有ライブラリは、しばしばプラグインと呼ばれる。

【 0 0 1 2 】

【発明が解決しようとする課題】

これまで、上述した耐タンパマイクロプロセッサ上で、共有ライブラリを使用できるようなアーキテクチャの提案は行なわれていない。

【 0 0 1 3 】

耐タンパマイクロプロセッサ上で共有ライブラリを実装するためには、以下に述べる要件を満たす必要がある。すなわち、アプリケーションプログラムから共有ライブラリのルーチンを呼び出すことが可能であること。および、ルーチンを呼び出す際にデータをルーチンに渡し、ルーチンから復帰するときも呼出元に処理結果のデータを返すことが出来ること、が要求される。

【 0 0 1 4 】

また上記に加えて、耐タンパマイクロプロセッサが提供するデータ等の保護機能を有効に機能させるためには、アプリケーションプログラムと共有ライブラリとの間でやりとりされる情報の秘密をOS等から守る必要がある。ルーチン呼出の際に秘匿したいデータをやりとりする場合、データを渡す前に相手が信用できるかどうか（OS等によって悪意のある別の共有ライブラリに置き換えられていないかどうか）を認証する必要がある。また、この認証が終った後で呼出先のルーチンがすり替えられることも防止しなければならない。共有ライブラリが複数のアプリケーションプログラムから同時に使用される場合は、共有ライブラリが1つのプログラムの秘密を他のプログラムに漏らすようなことはあってはならない。

【 0 0 1 5 】

また共有ライブラリは、任意のアプリケーションから使用可能でなければならない。つまり認証により、ある特定のアプリケーションプログラムからしか使用できないような共有ライブラリは、用途が限定されるため共有ライブラリとしては不十分である。一方、共有ライブラリを使用するアプリケーションプログラムの側は、共有ライブラリに秘匿にしたいデータを渡す前に、共有ライブラリがこのデータを他に漏らさないことを確認できることが望ましい。

【0016】

このためアプリケーションプログラムが、共有ライブラリを認証する手段が用意されていなければならない。

【0017】

共有ライブラリの動作は、アプリケーションプログラムからデータを受け取り、これに対しある処理をして、その結果をアプリケーションプログラムに返すことである。このとき、アプリケーションプログラムから受け取ったデータや、その処理結果を、アプリケーションプログラムと共有ライブラリ以外の第三者に漏洩してはならない。すなわち、データの受渡しを暗号化して行なわなければならないだけでなく、受け取ったデータに対応する処理結果を返す先のプログラムが、元のデータを提供したアプリケーションプログラムと同一であることを確かめる必要もある。

【0018】

また、誰でもこの共有ライブラリを使うアプリケーションプログラムを書くことができるため、悪意のあるアプリケーションプログラムから使用される可能性がある。その場合でも共有ライブラリの内部動作を解析から防御できなければならない。つまり、共有ライブラリの実行コードをアプリケーションプログラムが読み出したり、共有ライブラリに渡されたデータが処理される途中過程のデータをアプリケーションプログラムが盗み見たりすることを、防止できなければならない。

【0019】

本発明は従来の耐タンパマイクロプロセッサでは使用することのできなかった共有ライブラリを使用可能にすること。および、上記要件を満たしつつ、かつ前

述した利点の得られる共有ライブラリの使用方法、およびそのプログラムを提供することを目的とする。

【 0 0 2 0 】

【課題を解決するための手段】

上記目的を解決する、本発明の第一の側面は以下を要旨とする。暗号化されたコードを復号化して実行する機能を有すると共に、少なくとも1つのプログラム及びそのプログラムが呼び出す少なくとも1つの共有ライブラリに対応した複数の暗号鍵を格納する複数の領域からなるテーブルを有するマイクロプロセッサ上で、呼出元プログラムから前記少なくとも1つの共有ライブラリを呼び出して使用する方法であって、まず、共有ライブラリ用のタスクを作成し、次にこのタスクにタスク識別子を割り当てる。次に共有ライブラリのヘッダから予め記録されている命令鍵を取得し、次にこの命令鍵を前記マイクロプロセッサ内の前記共有ライブラリに割り当てられた識別子に対応するテーブルの領域に格納する。次に共有ライブラリ内のローダーを実行して初期化を行ない、共有ライブラリ内のエントリポイントを経由してアプリケーションへ制御を返す。

【 0 0 2 1 】

このようにして実現された共有ライブラリの使用方法にあつては、共有ライブラリそのものを一つの識別子を有するタスクとして処理し、かつこの識別子に対応するプロセッサ内の場所に共有ライブラリを暗号・復号化する為の命令鍵を記録するので、共有ライブラリの実行コードが保護される。

【 0 0 2 2 】

また本発明における第二の側面は、暗号化されたコードを復号化して実行する機能を有すると共に、少なくとも1つのプログラム及びそのプログラムが呼び出す少なくとも1つの共有ライブラリに対応した複数の暗号鍵を格納する複数の領域からなるテーブルを有するマイクロプロセッサ上で、呼出元プログラムから前記少なくとも1つの共有ライブラリを呼び出して使用する方法であって、以下を要旨とする。まず共有ライブラリ用のタスクを作成し、このタスクにタスク識別子を割り当てる。次に、共有ライブラリのヘッダから、予め記録されている命令鍵を取得し、この命令鍵を前記マイクロプロセッサ内の前記共有ライブラリに割

り当てられた識別子に対応するテーブルの領域に格納する。共有ライブラリ内のローダーを実行して初期化を行ない、上記ローダーで初期化をする際にこの共有ライブラリが使用するデータを暗号化するためのデータ鍵を作成する。次いでこのデータ鍵を、上記プロセッサ内の当該共有ライブラリに対応する識別子が割り当てられたの前記テーブルの領域に格納し、共有ライブラリ内のエントリポイントを経由してアプリケーションへ制御を返し、呼出元からの呼出を待機する。呼出元からの呼出後で呼出元と認証を行う。呼出元側でこの認証情報を使用して認証を行ない、認証が正しく行なわれた後呼出元との間で使用する共通鍵を作成する。呼出元が作成した共有メモリ領域のアドレスを受け取り、上記共有メモリ領域を呼出元とのデータ交換に使用する共有暗号化データ領域に設定する。そして、呼出元から共有ライブラリ内のサブルーチン呼出信号を受け取り、呼出元のチェックサムを検証する。呼出元とのチェックサムが共有ライブラリのチェックサムと一致した場合に要求された処理を行ない、上記処理の結果を前記共有暗号化データ領域にチェックサムを加えて送付する。

【 0 0 2 3 】

このような共有ライブラリでは、上記第一の側面に加えてプログラムから要求された処理を共有ライブラリが行う際に、共有ライブラリのデータ鍵を使用して処理内容および処理結果のデータを暗号化する。このため、共有ライブラリ自身のコードの暗号化に加えて、内部処理方法や処理結果が外部へ漏洩することを防げる。また、処理の要求を受ける時と処理結果を呼出元に返す時に互いのチェックサムが一致するかどうかを検証するため、処理の前後で呼出元がOSなどによって変更された場合にも、共有ライブラリ内の処理内容や処理結果を確実に保護できる。また加えて、呼出元の個数だけロードを行うため、一つの共有ライブラリを複数の呼出元から使用することも可能である。

【 0 0 2 4 】

【発明の実施の形態】

以下、図面を参照しながら本発明の一実施の形態を説明する。

【 0 0 2 5 】

図1は、本発明が適用されるマイクロプロセッサ（耐タンパマイクロプロセッ

サ) 1の基本ハードウェア構成図である。マイクロプロセッサ1は、プロセッサ・コア10と、コード・データ暗号／復号処理部11と、鍵値テーブル13と、タスクID格納レジスタ14と、乱数生成部15と、外部バスインタフェース12とを有する。

【0026】

耐タンパマイクロプロセッサとして使用されるマイクロプロセッサ1が、通常のマイクロプロセッサと大きく異なる点は、その内部にコード・データ暗号／復号処理部11を有している点である。バス上の実行コードやデータは、暗号化された状態で流れる。プロセッサ・コア10に入力される実行コードやデータは、コード・データ暗号／復号処理部11によって復号化されて入力される。またデータがプロセッサ・コア10から再度バス上へ流れて出るときには、コード・データ暗号／復号処理部11を経由して暗号化されて出力される。コード・データ暗号／復号処理部11は暗号化や復号化に暗号鍵を使用する。この暗号鍵は、同プロセッサ1内の鍵値テーブル13から取得する。また、鍵値テーブル13の中から使うべき暗号鍵を選択・取得するために、タスクID格納レジスタ14を使う。

【0027】

図2は、鍵値テーブル13の構成を示す図である。鍵値テーブル13は、暗号鍵の値を格納するレジスタからなる鍵値テーブルエントリ130を複数個並べたものである。この鍵値テーブルエントリは、後述する0～n番までのn+1個からなるタスクIDの数だけ設けることが可能である。各鍵値テーブルエントリ130は、タスクIDごとに130₀、130₁、130₂、・・・、130_nのように互いに識別される。鍵値テーブルエントリ130_i (0≤i≤n) には、後述するプログラム3の実行コード32を暗号・復号化するための鍵である命令鍵k1、プログラム3が操作するデータを暗号化・復号化するためのデータ鍵k2、命令鍵k1やデータ鍵k2が適用されるメモリ範囲を示すアドレス情報13aを格納することができる。鍵値テーブル13には複数の鍵値テーブルエントリ130₀、130₁、130₂、・・・、130_nが並んでいるため、それぞれの鍵値テーブル130_i (0≤i≤n) に別々のプログラム3の命令鍵を格納するこ

とができる。これにより異なる暗号鍵で暗号化された複数のプログラムのインスタンス（プロセス等）を疑似並列動作させるマルチタスク動作に対応可能となる。

【0028】

タスクID格納レジスタ14は、タスクIDを1つ格納するレジスタである。単一スレッドのプロセスや、複数スレッドを使うプロセスの中の各スレッド、又は呼び出された共有ライブラリのインスタンス（以下これらのプロセス、スレッドやインスタンスを、タスクと呼ぶ）をマルチタスク動作で疑似並列動作させているとき、現在マイクロプロセッサが実行中のタスクを識別するために、タスクID格納レジスタ14を用いる。

【0029】

乱数生成部15は、プロセッサ・コア10からの読み出しに対して、毎回異なる乱数を提供する。この乱数生成部15はプログラムが暗号化を行うための鍵や、認証のために必要な乱数を生成するために使用することができる。

【0030】

本発明が適用されるマイクロプロセッサ上で動作するソフトウェアは、システムソフトウェア（OS）と、アプリケーションプログラム3（以下、単にプログラムと呼ぶ）と、共有ライブラリ4からなる。

【0031】

以下では、アプリケーション3と、共有ライブラリ4の構成等について説明する。図3は、本発明に係るプログラム3の構成図である。プログラム3はヘッダ31と、実行コード32と、初期化済みデータ33と、インポートテーブル34からなる。ヘッダ31には、プログラム3の実行コード32を復号化するための命令鍵k31が含まれる。インポートテーブル34は、プログラム3が使用する共有ライブラリ4や、それに含まれるシンボル（シンボルは後述する共有ライブラリ内の各サブルーチン等を識別するための識別子である）を指定するもので、共有ライブラリ4のロードをするために必要な情報である。

【0032】

図4は、本発明に係る共有ライブラリ4の構成図である。共有ライブラリ4は

プログラム3と同様、ヘッダ41と、実行コード42と、初期化済みデータ43と、インポートテーブル44からなる。共有ライブラリ4のインポートテーブル44は、共有ライブラリ4自身が別の共有ライブラリを使用する場合に必要な情報を含む。共有ライブラリ4の実行コード42はさらに、ブートストラップルーチン42aと、エントリポイントコード42bと、それぞれ機能を持ったいくつかのサブルーチン42cとからなる。図4では、複数のサブルーチンを添字 i ($1 \leq i \leq n$)で、 $42c_1$ 、 $42c_2$ 、 \dots 、 $42c_n$ のように区別してある。

【0033】

ブートストラップルーチン（ローダー）42aは、共有ライブラリ4が呼出元のプログラム3等によってロードされる時に必要な処理を行うためのものである。エントリポイントコード42bは、呼出元のプログラム3等から共有ライブラリ4が使用されるときの呼出先となる共有ライブラリ4のプログラム3に対する入口となる部分である。共有ライブラリ4のその他のサブルーチン $42c_i$ ($1 \leq i \leq n$)が、実際にプログラム3が使用する機能を実装したコードである。共有ライブラリ4の各サブルーチン $42c_i$ ($1 \leq i \leq n$)は、それぞれを共有ライブラリ4の中で識別できるような識別子が付与されている。

【0034】

また共有ライブラリの実行コード42内には、共有ライブラリ4が、後述するプログラム3等からの呼出に応じて処理を行う際に、要求された処理を行う際に使用するデータを暗号化するためのデータ鍵 k_{42} も含んでいる。

【0035】

（タスクの開始）

次に図5を参照しながら、OSがユーザからのプログラム実行の要求や、既存プロセスによる新規プロセスや新規スレッド作成の要求により、タスクの実行を開始する手順を説明する。

【0036】

OSは最初にタスクの作成を行う（ステップ51）。これはメモリの確保や、タスクを管理するためのデータ構造の作成等を含む。また、OSは新しいタスクのためのタスクIDの割当も行う。

【0037】

次にOSはプログラム3のヘッダ31より命令鍵k31を取得し、先ほど割り当てたタスクIDに対応する鍵値テーブルのエントリ130_i($0 \leq i \leq n$)に命令鍵k31を格納する(ステップ52)。

【0038】

また、OSはプログラム3のインポートテーブル34を参照し、ここに記述された各共有ライブラリ4を、後述する手段(図6参照)によりロードする(ステップ53)。インポートテーブル34には任意の数の共有ライブラリ4が記述されている可能性がある。従って、OSは記述された共有ライブラリ4それぞれに関して、別個にロードの作業を行う。なお、タスク開始前に共有ライブラリ4をロードするかわりに、タスクが開始された後でタスクからの要求に応じて共有ライブラリ4をロードしても良い。

【0039】

OSは要求された全ての共有ライブラリ4のロードが完了すると、新しいタスクにコンテキスト(ただし、ここでコンテキストとは、プロセスの処理状況や環境が記載されているデータを指す)の切り替えを行ない、タスクの実行を開始する(ステップ54)。このタスク切り替えの際にプログラムのタスクIDがタスクID格納レジスタ14に格納される。

【0040】

耐タンパマイクロプロセッサ1において、プログラムの実行コード32は次のように実行される。外部バスインタフェース12が外部メモリより命令コードを取得すると、コード・データ暗号/復号処理部11は、タスクID格納レジスタ14に格納されたタスクIDに対応する鍵値テーブル13の内容を参照し、ここに格納されたプログラム3の命令鍵k31によって実行コード32を復号化する。次に復号化された実行コード32はプロセッサ・コア10に渡されて実行される。

【0041】

(共有ライブラリのロード)

図6は、OSが共有ライブラリをロードする手順を示す図である。

【0042】

通常のマイクロプロセッサでは、共有ライブラリのために呼出元のプログラムとは別のタスクを作成するようなことはしない。しかし耐タンパプロセッサ 1 では、プログラム 3 又は共有ライブラリ 4 の一方に悪意があった場合の安全性を考慮し、共有ライブラリ 4 にも別のタスクを作成する。このため、OS が共有ライブラリ 4 をロードするときに最初に行う作業は、タスクの作成及び新しいタスクのためのタスク ID の割り当てである（ステップ 6 1）。

【 0 0 4 3 】

次に OS は共有ライブラリ 4 のヘッダ 4 1 より命令鍵 k_{41} を取得し、割り当てられたタスク ID に対応する鍵値テーブル 1 3 のエントリ $130_i (0 \leq i \leq n)$ にこの命令鍵 k_{41} を格納する（ステップ 6 2）。このステップにより予め命令鍵 k_{41} で暗号化されている共有ライブラリ 4 をマイクロプロセッサ 1 で復号化可能となるとともに、各命令鍵 k_{41} が別々のエントリ $130_i (0 \leq i \leq n)$ に格納される。従って、複数の共有ライブラリ 4 が存在する場合でも、これらを識別可能となり安全である。

【 0 0 4 4 】

また、OS は必要に応じて共有ライブラリ 4 のインポートテーブル 4 4 を参照し、ここに記述された他の共有ライブラリ 4 をロードする（ステップ 6 3）。このロードの手順は、プログラム 3 のインポートテーブル 3 4 に従って共有ライブラリ 4 をロードする場合の手順と同じである。

【 0 0 4 5 】

次に、OS は共有ライブラリ 4 のブートストラップルーチン（ローダー）4 2 a を実行する（ステップ 6 4）。ブートストラップルーチン 4 2 a は、必要な初期化の処理を行なった後、エントリポイントコード 4 2 b に制御を渡す（ステップ 6 5）。次いで、エントリポイントコード 4 2 b は、制御を OS に戻す。制御が OS に戻ると共有ライブラリ 4 は待機状態に移行する（ステップ 6 6）。

【 0 0 4 6 】

なお、ロードしようとしている共有ライブラリ 4 を、既に実行中の他のタスクが使用している場合であっても、共有ライブラリ 4 のロードは呼出元のタスク毎に毎回行う。結果として、呼出元タスクと同じ数だけのタスク ID が同じ共有ライ

ブラリ 4 に対し割り当てられる。ただし共有ライブラリ 4 の実行コード 4 2 は、OS やマイクロプロセッサ 1 の仮想メモリの仕組みを利用すれば、呼出元タスクが複数ある場合でも 1 回だけメモリにロードするだけで良い。

【0047】

(共有ライブラリ内のサブルーチンの呼出)

タスク実行の途中で、共有ライブラリ 4 のサブルーチン 4 2 $c_i (1 \leq i \leq n)$ を呼び出す必要があるときは、図 7 に示すシーケンスによってサブルーチン 4 2 $c_i (1 \leq i \leq n)$ 呼出の処理が行なわれる。

【0048】

図 8 は、サブルーチン呼出パラメータブロックの構造 8 である。呼出元のタスクは、呼出先の共有ライブラリ 4 の識別子 8 1 と、呼び出すサブルーチン 4 2 $c_i (1 \leq i \leq n)$ の識別子 8 2 と、サブルーチン 4 2 $c_i (1 \leq i \leq n)$ に渡すパラメータ 8 3 とを、サブルーチン呼出パラメータブロック 8 に格納する。サブルーチン呼出パラメータブロック 8 を作成した後、呼出元のタスクはシステムコールを使い、サブルーチン呼出パラメータブロック 8 を渡して、OS にサブルーチン呼出の要求を出す。

【0049】

OS はこの要求を受けると呼出元のタスクを停止させ、サブルーチン呼出パラメータブロック 8 の共有ライブラリ識別子 8 1 を参照して呼出先の共有ライブラリ 4 のタスクを選択し、その共有ライブラリ 4 にタスクの切り替えを行う。このとき OS はサブルーチン呼出パラメータブロック 8 を共有ライブラリのタスクに渡す。共有ライブラリ 4 にタスクの切り替えを行うと共有ライブラリ 4 のタスク ID がタスク ID 格納レジスタ 1 4 に格納される。するとこれまで待機状態であった共有ライブラリ 4 のエントリポイントコード 4 2 b の実行が再開される。

【0050】

共有ライブラリ 4 の暗号化された実行コード 4 2 を実行する手順は、上記で説明したプログラム実行コード 3 2 の実行手順と同じである。

【0051】

共有ライブラリ 4 のエントリポイントコード 4 2 b は、実行が再開されると渡

されたサブルーチン呼出パラメータブロック 8 の内容を参照し、ここに指定されたサブルーチン識別子 8 1 に対応するサブルーチン $42c_i (1 \leq i \leq n)$ を呼び出す。呼び出されたサブルーチン $42c_i (1 \leq i \leq n)$ はサブルーチン呼出パラメータブロック 8 にあるパラメータ 8 3 等を参照し、要求された処理を行う。処理の結果として返すべきデータはサブルーチン呼出パラメータブロック 8 に格納し、処理が完了するとエントリポイントコードに復帰し、次にここから OS に復帰する。OS に処理が復帰すると共有ライブラリ 4 は再び待機状態に戻り、OS はサブルーチン呼出パラメータブロック 8 を呼出元のタスクに返して呼出元タスクの実行を再開する。

【0052】

(マルチスレッド動作)

共有ライブラリ 4 の呼出元であるのプログラム 3 がマルチスレッド動作をする場合、複数のスレッドが同時に同じ共有ライブラリ 4 のタスクを使用することはできない。

【0053】

そこで、OS は呼出元プロセスが実行中に新しいスレッドの作成を要求すると、呼出元プログラム 3 が使う全ての共有ライブラリ 4 に関して、もう 1 度ロードの処理を行う。結果として、各共有ライブラリ 4 には、それぞれ呼出元プロセスのスレッドの数と同じだけの数のタスク ID が割り当てられる。OS はあるスレッドから共有ライブラリ 4 のサブルーチン $42c_i (1 \leq i \leq n)$ 呼出を要求されると、その共有ライブラリ 4 に割り当てられた未使用のタスク ID を選択して、このタスク ID を使用して共有ライブラリ 4 のサブルーチン $42c_i (1 \leq i \leq n)$ を呼び出す。

【0054】

また、OS は使用するタスク ID の数を節約するため、共有ライブラリ 4 のロードをタスクの作成時ではなく、サブルーチン $42c_i (1 \leq i \leq n)$ 呼出のときにタスク ID が不足した場合に行うようにしても良い。

【0055】

(共有ライブラリに固有の秘匿データの保持方法)

共有ライブラリ 4 のサブルーチン $42c_i (1 \leq i \leq n)$ が、共有ライブラリ 4 内部

での処理経過や処理方法等の秘匿データを保持するには次のようにする。

【 0 0 5 6 】

共有ライブラリ 4 の作成者は、秘匿データ暗号化のための暗号化鍵であるデータ鍵 k_{42} を 1 つ作り、実行コード 4 2 の中に埋め込む（図 4 参照）。このデータ鍵 k_{42} の値は実行コード 4 2 の中に埋め込まれており、実行コード 4 2 は命令鍵 k_{41} によって暗号化されている。従って、共有ライブラリ 4 の命令鍵 k_{41} を知らない者はこのデータ鍵 k_{42} を共有ライブラリ 4 から取り出すことはできない。秘匿すべきデータを暗号化するには、このデータ鍵 k_{42} の値と、暗号化すべきメモリ領域のアドレスを、鍵値テーブル 1 3 に格納する。すると指定されたメモリ領域に対して読み書きされるデータは、コード・データ暗号／復号処理部 1 1 により暗号化／復号化される。また、このデータ鍵 k_{42} を使って予め暗号化したデータを共有ライブラリ 4 の初期化済みデータ 4 3 として配布し、共有ライブラリ 4 のサブルーチン $42c_{i(1 \leq i \leq n)}$ がこれを上記のように復号化して使っても良い。

【 0 0 5 7 】

データを上述のメモリ領域に書き込んでいったん呼出元に復帰した後、再び呼び出されたときにこのデータを読み出す必要がある場合は、同じデータ鍵 k_{42} を鍵値テーブル 1 3 に再度格納すれば良い。

【 0 0 5 8 】

ただし上記の方法では、同じ共有ライブラリ 4 を使う限り、何回ロードしても毎回同じデータ鍵 k_{42} の値を使うことになる。つまり、同じデータ鍵 k_{42} を使うと、1 回のプログラム実行時の暗号化されたデータを保存しておいて、もう 1 度プログラムを実行した時にこのデータを同じメモリ領域に書き込み、前回実行したときの状態を再現することができる。このようにデータの再利用ができてしまうと、共有ライブラリ 4 の動作に対する攻撃手段として使われる恐れがあり、不都合な場合がある。

【 0 0 5 9 】

そのような場合は、白川他による特願 2000-333635 に記載のように、命令鍵とデータ鍵を鍵のペアとみなし、データ鍵 k_{42} としてプロセッサの乱数生成部 1

5で生成した乱数を使用すればよい。共有ライブラリ4のタスクは、実行時に乱数生成部15より乱数を取得し、これをデータ鍵 k_{42} として、このデータ鍵 k_{42} を使用して暗号化すべきメモリ領域のアドレスとともに、鍵値テーブル13に格納する。共有ライブラリ4のタスク毎に乱数生成部15で生成したデータ鍵 k_{42} を使用すれば、異なるタスクでは異なる値のデータ鍵 k_{42} が使われる。従って、上記のようにして得られたデータ鍵 k_{42} を使って暗号化されたデータは、再利用することができなくなる。

【0060】

命令鍵 k_{41} とデータ鍵 k_{42} のペアは呼出元に復帰しても内容が失われないため、再び共有ライブラリ4のルーチンが呼び出されても、このデータ鍵 k_{42} で暗号化したデータは読み出すことができる。

【0061】

(データの受渡し)

共有ライブラリ4とその呼出元(プログラムないしは、その他の共有ライブラリ)は、上記のサブルーチン呼出パラメータブロック8を用いて互いにデータをやりとりすることができる。しかし、このサブルーチン呼出パラメータブロック8の内容は暗号化されていない。そのため、オペレーティングシステムにより内容が見られてしまう恐れがある。

【0062】

共有ライブラリ4とその呼出元が秘匿データを互いにやりとりするためには、寺本他による特願2000-402672で提案されているように、両者の間でDiffie-Hellman鍵交換シーケンスを用いて鍵の生成を行なえばよい。Diffie-Hellman 鍵交換シーケンスにより生成される鍵は、鍵交換を行なった2者のみが計算できる。すなわち、両者のやりとりを観察することのできるOSであっても、この鍵の値を知ることとはできない。以下、この鍵を共通鍵 c_k と呼ぶことにする。

【0063】

秘匿データ受渡しのための領域は、OSが提供するメモリ共有手段により、両者の間で共有されるメモリ領域を割り当てる。この領域に対し、共有ライブラリ4と呼出元がそれぞれのタスクの鍵値テーブルエントリ $130_i (0 \leq i \leq n)$ に共通鍵

c k を登録すれば、相手がこのメモリ領域に暗号化して書き込んだ内容を互いに復号化して読むことができる。このようにして用意したメモリ領域を、以下では共有暗号化データ領域と呼ぶ。

【 0 0 6 4 】

共有ライブラリ 4 やその呼出元が相互に、相手が意図したプログラム 3 又は共有ライブラリ 4 であることを認証する必要がある場合は、上記の鍵交換シーケンスのメッセージに公開鍵方式による署名を与えることにより行う。例えば、プログラム 3 が共有ライブラリ 4 を認証する必要がある場合は、共有ライブラリ 4 に公開鍵と秘密鍵の組を予め与えておき、プログラム 3 の作成者に共有ライブラリ 4 の公開鍵を配布しておく。認証は、鍵交換のために共有ライブラリ 4 がプログラム 3 に送るメッセージに、共有ライブラリ 4 の秘密鍵による署名を付加し、プログラム 3 が共有ライブラリ 4 の公開鍵を用いてこの署名を検証する。反対に共有ライブラリ 4 がプログラム 3 を認証する必要がある場合は、プログラム 3 に公開鍵と秘密鍵のペアを与えておき、共有ライブラリ 4 の作成者にプログラム 3 の公開鍵を配布しておく。認証は、鍵交換のためにプログラムが共有ライブラリ 4 に送るメッセージに、プログラム 3 の秘密鍵による署名を付加し、共有ライブラリ 4 がプログラム 3 の公開鍵を用いてこの署名を検証する。なお、これらの方式では認証を鍵交換と同時に行うが、もしこれを別々に行なってしまうと、認証をしている相手と鍵交換をしている相手が同一であることを確かめる方法がないため、悪意のあるプログラムによる相手のなりすましが可能になってしまい、認証が正しく行なわれない。

【 0 0 6 5 】

(共有ライブラリの動作)

以上で説明した各部位の基本動作を用いて、本発明の一実施形態における実際の共有ライブラリ 4 の動作を説明する。ここで説明するプログラム 3 が、共有ライブラリ 4 を使用する手順における、共有ライブラリ 4 側の動作を図 9 に示す。またその時の、プログラム 3 側の動作を図 1 0 に示す。なお、図 1 1 は、そのときのプログラム 3 と共有ライブラリ 4 それぞれのメモリアドレス空間の配置である。

【 0 0 6 6 】

ここで説明する共有ライブラリ 4 は、プログラム 3 からデータを受け取ってある処理を行ないその結果を返す、という動作のルーチンを提供する。その他に、Diffie-Hellman 鍵交換シーケンスを行うための補助的なルーチンもプログラムから利用可能である。また、プログラム 3 の側から共有ライブラリ 4 を認証することができるよう、共有ライブラリ 4 には前述の認証用の公開鍵と秘密鍵が与えられているものとする。これらのうち、公開鍵の方は共有ライブラリ 4 と共に配布され、この共有ライブラリ 4 を使いたいプログラマは、この公開鍵を自分が作るプログラムに組み込むことができる。

【 0 0 6 7 】

共有ライブラリ 4 を使用するプログラム 3 が実行されると、先に図 6 を用いて説明したように、共有ライブラリ 4 のロードが行なわれる（ステップ 9 1）。ロードの際に共有ライブラリ 4 のブートストラップルーチン 4 2 a（図 6 参照）が実行される。この際マイクロプロセッサ 1 の乱数生成部 1 5 より乱数を取得してこれよりデータ鍵 k_{42} を生成し、これを鍵値テーブル 1 3 に共有ライブラリ 4 のデータ鍵 k_{42} として格納する。ロードが完了すると共有ライブラリ 4 は待機状態になり、プログラム 3 の実行が開始される。プログラム 3 も共有ライブラリ 4 と同様、自分が使うメモリ領域を暗号化するために乱数生成部 1 5 より乱数を取得し、データ鍵 k_{32} を生成して鍵値テーブル 1 3 にデータ鍵 k_{32} として格納する（ステップ 1 0 1）。

【 0 0 6 8 】

プログラム 3 は共有ライブラリ 4 の機能を使用する前に、データの受渡しのための共有暗号化データ領域 1 1 5 を準備する（ステップ 1 0 2）。次に共有ライブラリ 4 の Diffie-Hellman 鍵交換シーケンスを行うためのルーチンを呼び出す（ステップ 1 0 3）。またこれに伴い、共有ライブラリ 4 側も Diffie-Hellman 鍵交換シーケンスを実行する（ステップ 9 2）。鍵交換の際、共有ライブラリ 4 からプログラム 3 に送るメッセージには、共有ライブラリ 4 の認証用の秘密鍵を用いて署名を付加し、この認証情報をプログラムへ送付する（ステップ 9 3）。プログラム 3 は共有ライブラリ 4 からこの認証情報を受け取る（ステップ 1 0 4）。

ここで、共有ライブラリ 4 の認証用の公開鍵を用いてこの署名の検証を行う（ステップ 1 0 5）。この検証が成功した場合にのみ鍵交換シーケンスを終了させる（ステップ 1 0 6）。この署名の検証により、鍵交換を行なっている相手の共有ライブラリ 4 が、意図した正しい共有ライブラリ 4 であることを確かめることができる。

【 0 0 6 9 】

鍵交換の結果生成される共通鍵 c_k の値は、プログラム 3 の側ではプログラム 3 のデータ鍵 k_{32} により暗号化されてプログラム 3 以外から読み出せない秘匿データ領域 1 1 2 に書き込まれる。また同様に共有ライブラリ 4 の側でも上記共通鍵 c_k は、共有ライブラリ 4 のデータ鍵 k_{42} により暗号化され、共有ライブラリ 4 以外から読み出せない秘匿データ領域 1 1 4 に書き込まれる（ステップ 9 4）。次にプログラム 3 は、データ受渡しのための共有メモリ領域を OS に要求して確保し、共通鍵 c_k をこの共有メモリ領域に対し適用されるよう鍵値テーブル 1 3 に設定する（ステップ 1 0 7）。また、この共有メモリ領域のアドレスを共有ライブラリ 4 にも通知し、共有ライブラリ 4 の側でも共通鍵 c_k を同じメモリ領域に対して適用されるよう鍵値テーブル 1 3 に設定する。一方、共有ライブラリ 4 は、この共有メモリ領域のアドレスを受け取り、鍵の設定を行う（ステップ 9 5）。これによりこのメモリ領域は、共有暗号化データ領域 1 1 5 として使用できるようになる。

【 0 0 7 0 】

プログラム 3 が実際に目的の処理を行う共有ライブラリ 4 のサブルーチン 4 2 $c_{i(1 \leq i \leq n)}$ を呼び出すときは、共有暗号化データ領域 1 1 5 に渡すべきデータを入れ、またこのデータのチェックサム 1 1 1 を計算してデータに付加する。そして共有ライブラリ 4 のサブルーチン 4 2 $c_{i(1 \leq i \leq n)}$ に呼び出す（ステップ 1 0 9）。この呼出信号を受け取ると（ステップ 9 6）、共有ライブラリ 4 のサブルーチン 4 2 $c_{i(1 \leq i \leq n)}$ はまずチェックサム 1 1 1 を検査する（ステップ 9 7）。チェックサム 1 1 1 がデータ内容と一致しない場合はエラーとして終了する（ステップ 9 7 1）。チェックサム 1 1 1 が一致した場合、次に共有暗号化データ領域 1 1 5 からデータ及び呼出側からの要求を読みだす（ステップ 9 8）。こ

の処理の過程で作業用のメモリ領域が必要な場合は、共有暗号化データ領域 115 ではなく、共有ライブラリ 4 独自のデータ鍵 k_{42} により暗号化されたメモリ領域（作業領域）116 を作成して使用してもよい（ステップ 99）。次に、このデータに対し要求された処理を行う（ステップ 990）。要求された処理が完了したら、処理結果を共有暗号化データ領域 115 に格納し、チェックサム 111 を付加した後、プログラム 3 に復帰する（ステップ 991）。復帰するとプログラム 3 はチェックサム 111 を検査する（ステップ 110、ステップ 111）。これがデータ内容と一致する場合、返されたデータを使って処理を続行する S112。一致しない場合は、エラーを返し終了する（ステップ 113）。

【0071】

以上で説明した共有ライブラリ 4 は、鍵交換のための手続きやデータ受渡しの形式が公開されていれば誰でもこれを使うプログラム 3 を作ることができる。この共有ライブラリ 4 が安全性に関する要求を満たすことは次のように確認できる。

【0072】

プログラム 3 から渡されたデータの秘匿性は、プログラム 3 と共有ライブラリ 4 の間で共有される共有暗号化データ領域 115 が、共通鍵 c_k によって暗号化されていることにより保証される。

【0073】

ここで暗号化に用いている共通鍵 c_k は Diffie-Hellman 鍵交換により生成している。従って、どちらかが意図的に、あるいは誤ってこの共通鍵 c_k を開示してしまわない限り、この二者以外に共通鍵 c_k の値が知られることはない。また、処理結果を返す時も、同じ共有暗号化データ領域 115 を使うため、この内容は、プログラム 3 および共有ライブラリのみが閲覧可能である。もし処理結果を他のプログラムが読もうとしても、元の Diffie-Hellman 鍵交換による共通鍵 c_k の値を知らないので、復号化することはできない。

【0074】

共有ライブラリ 4 が処理を行なっている途中で、OS が呼出元のプログラム 3 を別のプログラム 3 にすり替えて、置き換えた後のプログラム 3 が共有ライブラリ

4 から返された処理結果を盗み取るという攻撃が考えられる。しかし、呼出元のプログラムは、すり替えの前後で違う命令鍵 k_1 (k_{31}) のプログラム 3 になるため、鍵値テーブル 1 3 でペアとして管理されている命令鍵 k_1 とデータ鍵 k_2 (k_{32}) の内容は共に変化する。共通鍵 c_k はすり替え前のデータ鍵 k_2 (k_{32}) で暗号化された秘匿データ領域 1 1 5 に格納されているが、すり替え後は鍵値テーブル 1 3 にあるデータ鍵 k_2 (k_{32}) の値が変わってしまうため、すり替え後のプログラム 3 は共通鍵 c_k を読むことができない。このため共有ライブラリ 4 より返されたデータをすり替え後のプログラム 3 が復号化することは原理的に不可能である。

【 0 0 7 5 】

別の攻撃手段として、プログラム 3 と共有ライブラリ 4 との間のデータ受渡しの途中過程で、他のプログラム 3 がこの内容を改変することにより誤動作を誘導することが考えられる。しかし、改変をする側のプログラム 3 はこのデータの暗号化に使われている共通鍵 c_k の値を知らないため、改変結果を復号化した結果を予測することはできない。改変を行うとデータを復号化した結果はでたらめな内容になり、チェックサム 1 1 1 による整合性の検査で改変が検出される。

【 0 0 7 6 】

複数のタスクが同時に同じ共有ライブラリ 4 を使用している場合でも、各呼出元のタスク毎に別のタスクを共有ライブラリ 4 のために割り当てるので、各共有ライブラリ 4 はそれぞれ別のデータ鍵 k_{42} を使うことになる。このため、共有ライブラリ 4 がある呼出元から受け取ったデータを、誤って他の呼出元に渡してしまうようなことは発生しない。

【 0 0 7 7 】

共有ライブラリ 4 を呼び出すプログラム 3 自体が、共有ライブラリ 4 の実行コード 4 2 を見ることは、共有ライブラリ 4 がそれに固有の命令鍵 k_{41} により暗号化されているため、不可能である。また、共有ライブラリ 4 が処理の過程で作業用に使うメモリの内容も、共有ライブラリ 4 にしか知られないデータ鍵 k_{42} により暗号化された秘匿データ領域 1 1 4 を使うため、これを覗き見ることはできない。

【 0 0 7 8 】

なお、上記の共有ライブラリ 4 は予め認証用の秘密鍵を持っている。このため、この秘密鍵を知らない第三者は、この共有ライブラリ 4 の代替となる共有ライブラリ 4 を勝手に作ることはできない。これは動的リンク共有ライブラリの利点である、既存の共有ライブラリと互換性のある共有ライブラリを自由に作ることができるという特徴に反する。しかし、これは共有ライブラリ 4 の動作の安全性をプログラム 3 に対して保証する上で必要不可欠な制約事項である。もし共有ライブラリ 4 の、元々の提供者以外の開発者が、機能追加等のためにこの共有ライブラリ 4 と互換性のある新しい共有ライブラリ 4 を作成する必要がある場合は、元の共有ライブラリ 4 の提供者に新しい共有ライブラリ 4 の安全性を確認してもらい、秘密鍵を埋め込んでもらうよう依頼すれば良い。

【 0 0 7 9 】

【発明の効果】

上記説明したように、本発明によれば耐タンパマイクロプロセッサ上で動作する、保護されたアプリケーションプログラムから、保護された共有ライブラリを使用することが可能となる。共有ライブラリを使うことにより、プログラムの開発効率や機能拡張性の向上を得ることができる。また、秘匿されたデータのやりとりや、相互認証を可能とすることにより、暗号化されたプログラムや共有ライブラリ内の処理方法や、処理結果などのデータの秘密を守ることが可能である。

【図面の簡単な説明】

【図 1】

本発明による共有ライブラリ使用の前提となるマイクロプロセッサの構成図。

【図 2】

図 1 のマイクロプロセッサ内部における、鍵値テーブルの構成図。

【図 3】

アプリケーションプログラムの構成図。

【図 4】

共有ライブラリの構成図。

【図 5】

本発明によるタスク実行開始の手順を示すフローチャート。

【図 6】

本発明による共有ライブラリをロードする手順を示すフローチャート。

【図 7】

本発明による共有ライブラリのルーチン呼出手続きを示すシーケンス図。

【図 8】

本発明によるサブルーチン呼出パラメータブロックの構成図。

【図 9】

プログラムから共有ライブラリを使用する際の、共有ライブラリ側のフローチャート。

【図 1 0】

プログラムから共有ライブラリを使用する際の、プログラム側のフローチャート。

【図 1 1】

プログラムと共有ライブラリのメモリアドレス空間の配置図。

【符号の説明】

- 1 耐タンパマイクロプロセッサ
- 1 0 プロセッサ・コア
- 1 1 コード・データ暗号/復号処理部
- 1 2 外部バスインタフェース
- 1 3 鍵値テーブル
- 1 4 タスク I D 格納レジスタ
- 1 5 乱数生成部
- k 1 命令鍵
- k 2 データ鍵
- 1 3 a アドレス情報
- 1 3 0 ₀ 鍵値テーブルエントリ
- 1 3 0 ₁ 鍵値テーブルエントリ

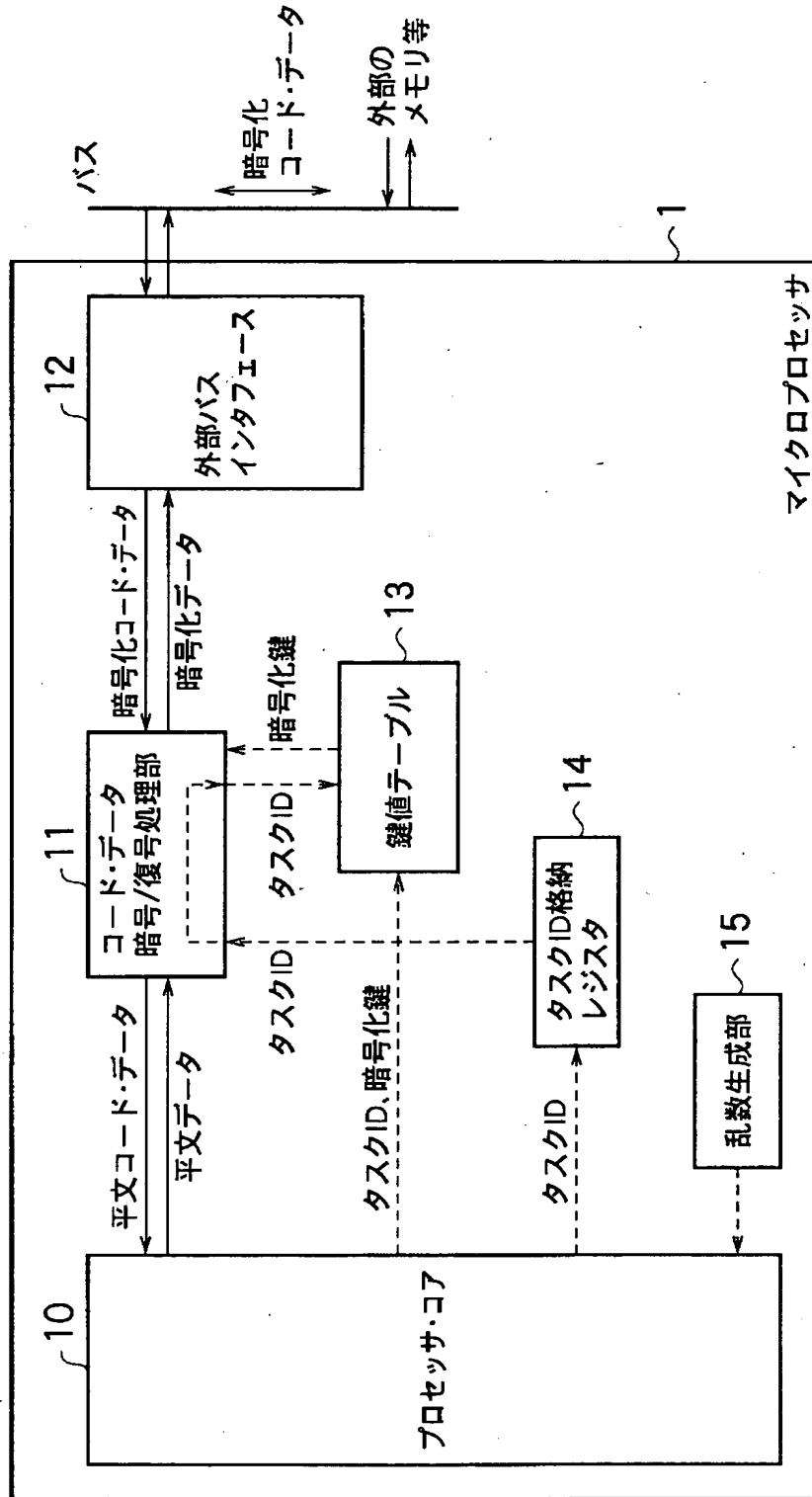
- 1 3 0 ₂ 鍵値テーブルエントリ
- 1 3 0 _n 鍵値テーブルエントリ
- 3 アプリケーションプログラム
- 3 1 ヘッダ
- k 3 1 命令鍵
- 3 2 実行コード
- 3 3 初期化済みデータ
- 3 4 インポートテーブル
- 4 共有ライブラリ
- 4 1 ヘッダ
- k 4 1 命令鍵
- 4 2 実行データ
- 4 2 a ブートストラップルーチン
- 4 2 b エントリポイントコード
- 4 2 c ₁ サブルーチン
- 4 2 c ₂ サブルーチン
- 4 2 c _n サブルーチン
- k 4 2 データ鍵
- 4 3 初期化済みデータ
- 4 4 インポートテーブル
- 8 サブルーチン呼出パラメータブロック
- 8 1 共有ライブラリ識別子
- 8 2 サブルーチン識別子
- 8 3 パラメータ
- 1 1 2 秘匿データ領域
- c k 共通鍵
- 1 1 5 共有暗号化データ領域
- 1 1 0 受け渡しデータ
- 1 1 1 チェックサム

1 1 4 秘匿データ領域

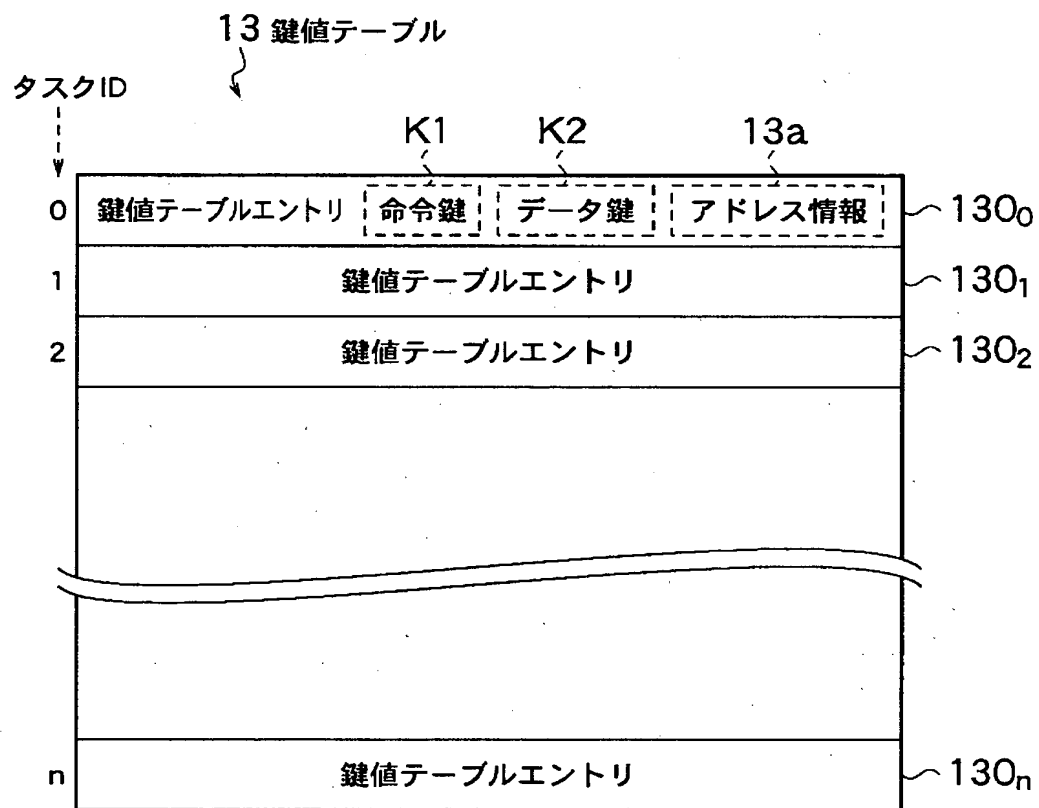
1 1 6 作業領域

【書類名】 図面

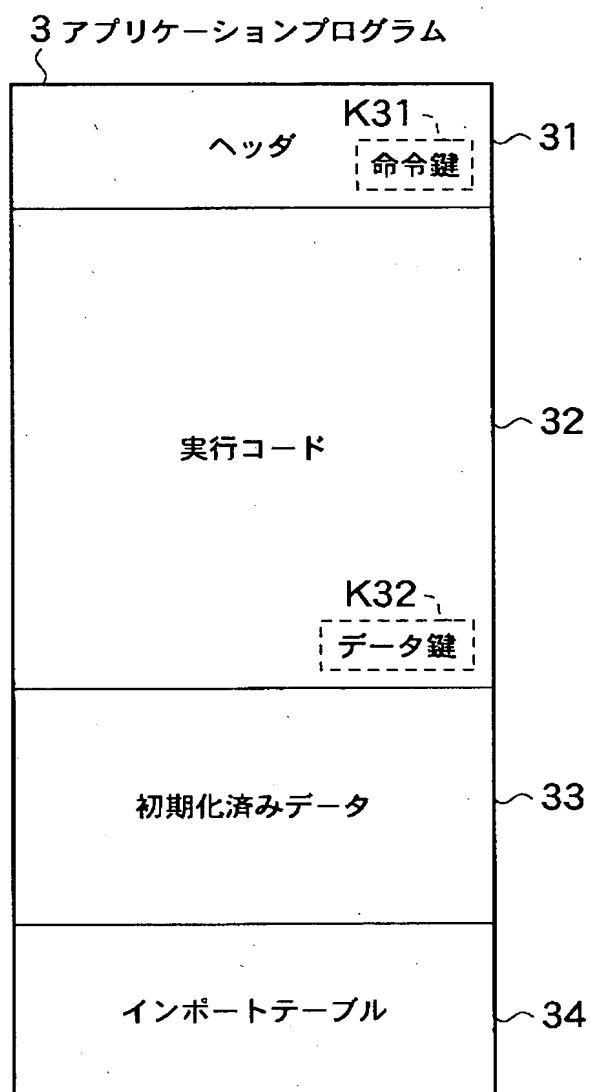
【図 1】



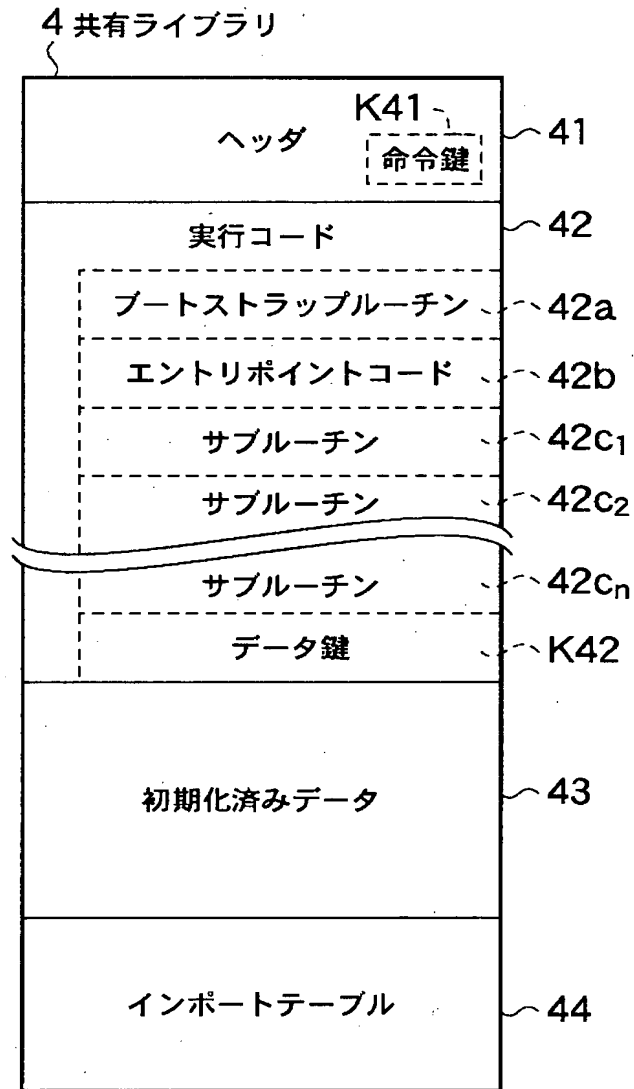
【図 2】



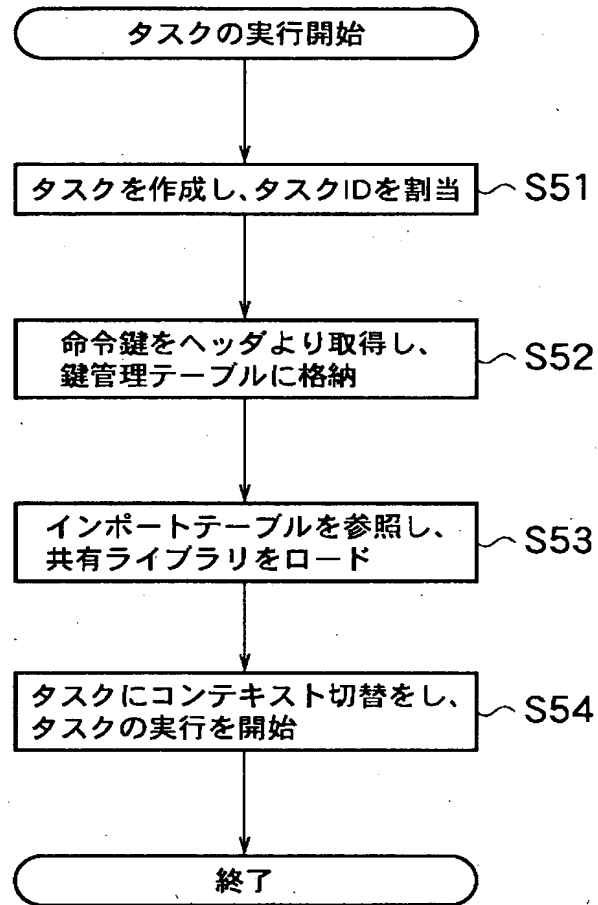
【図 3】



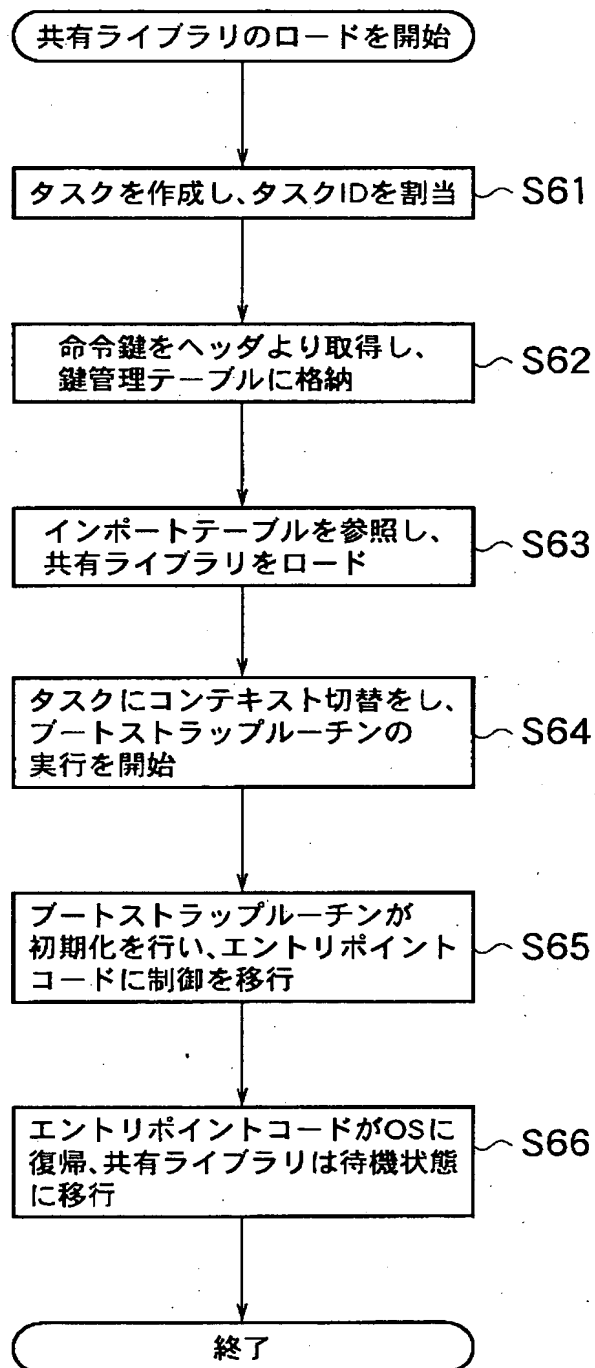
【図 4】



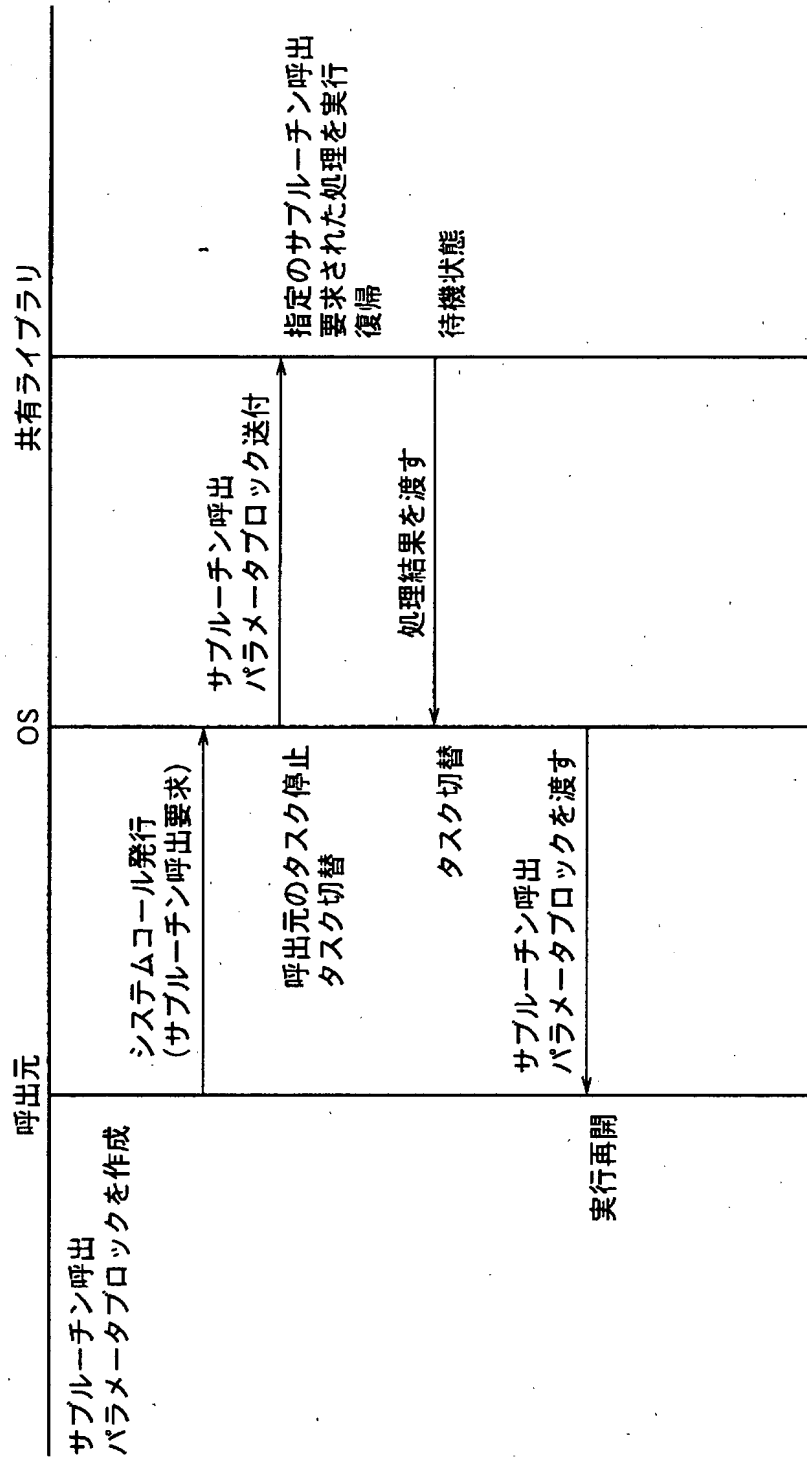
【図 5】



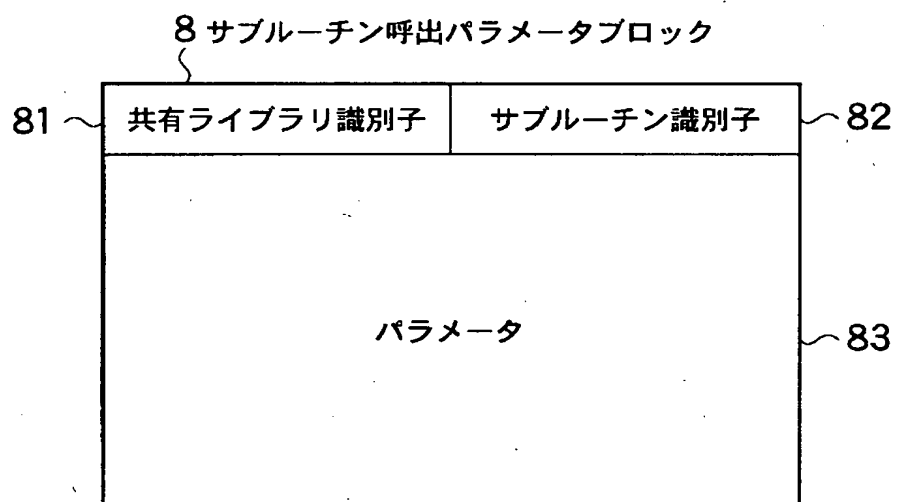
【図 6】



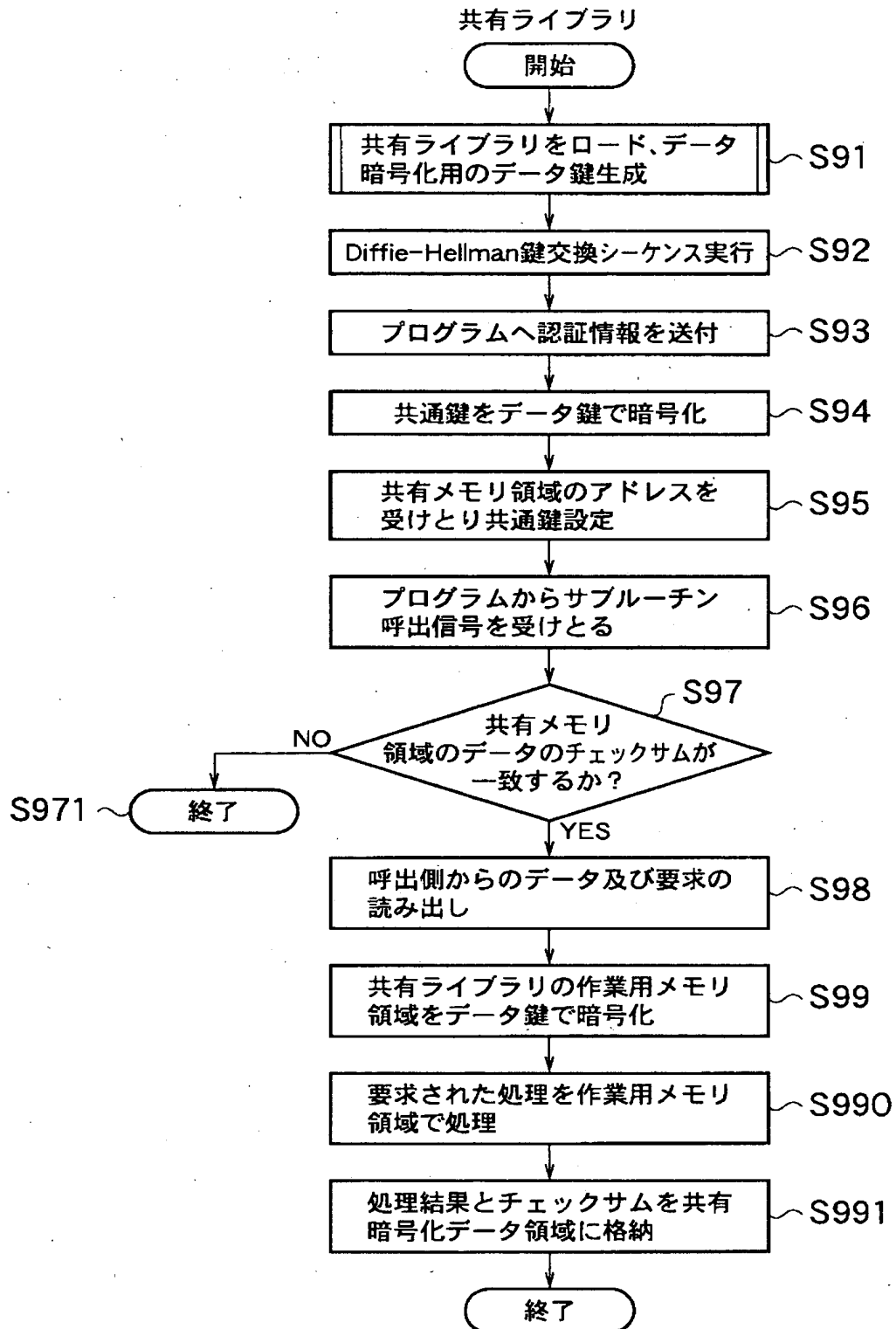
【図 7】



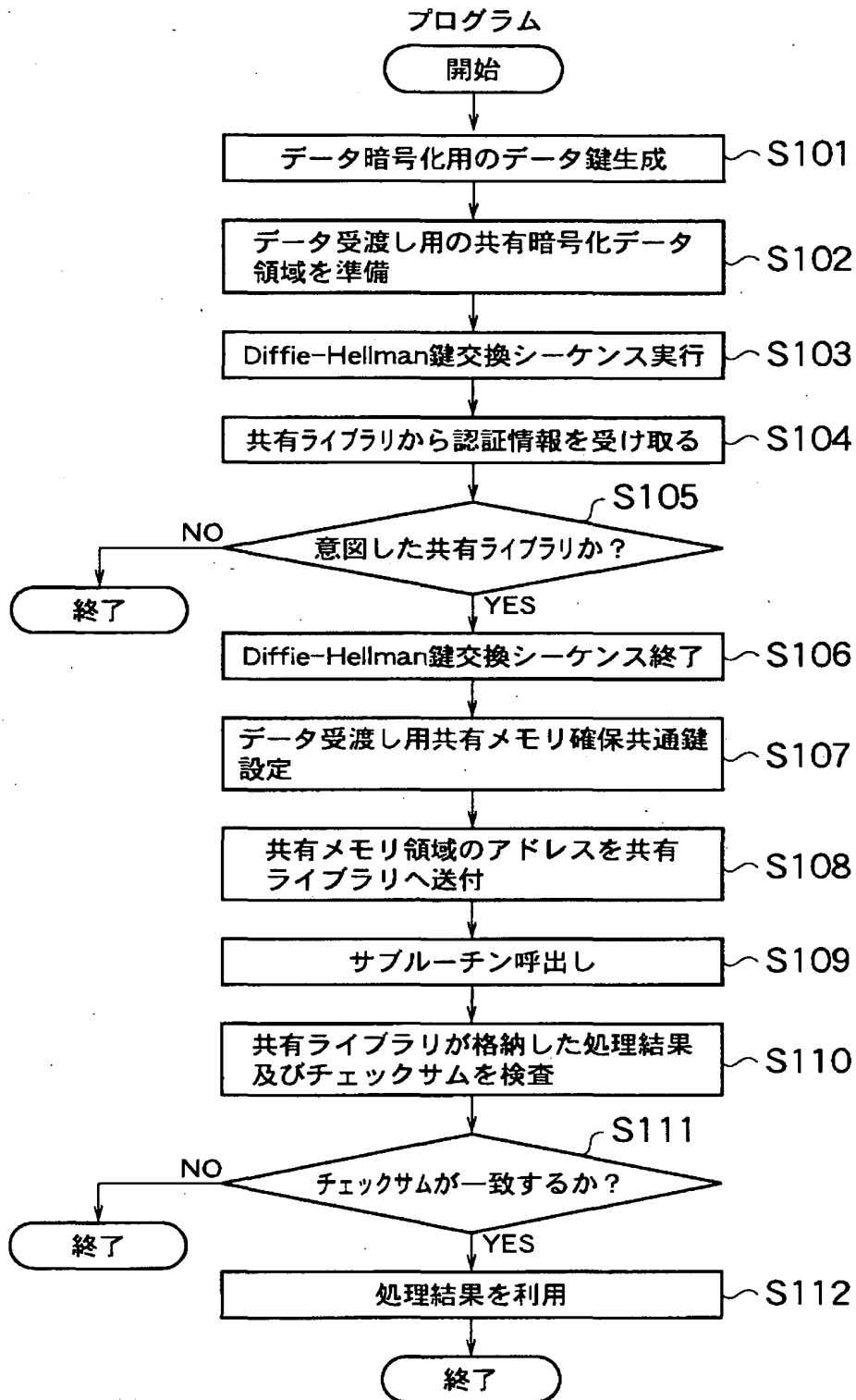
【図 8】



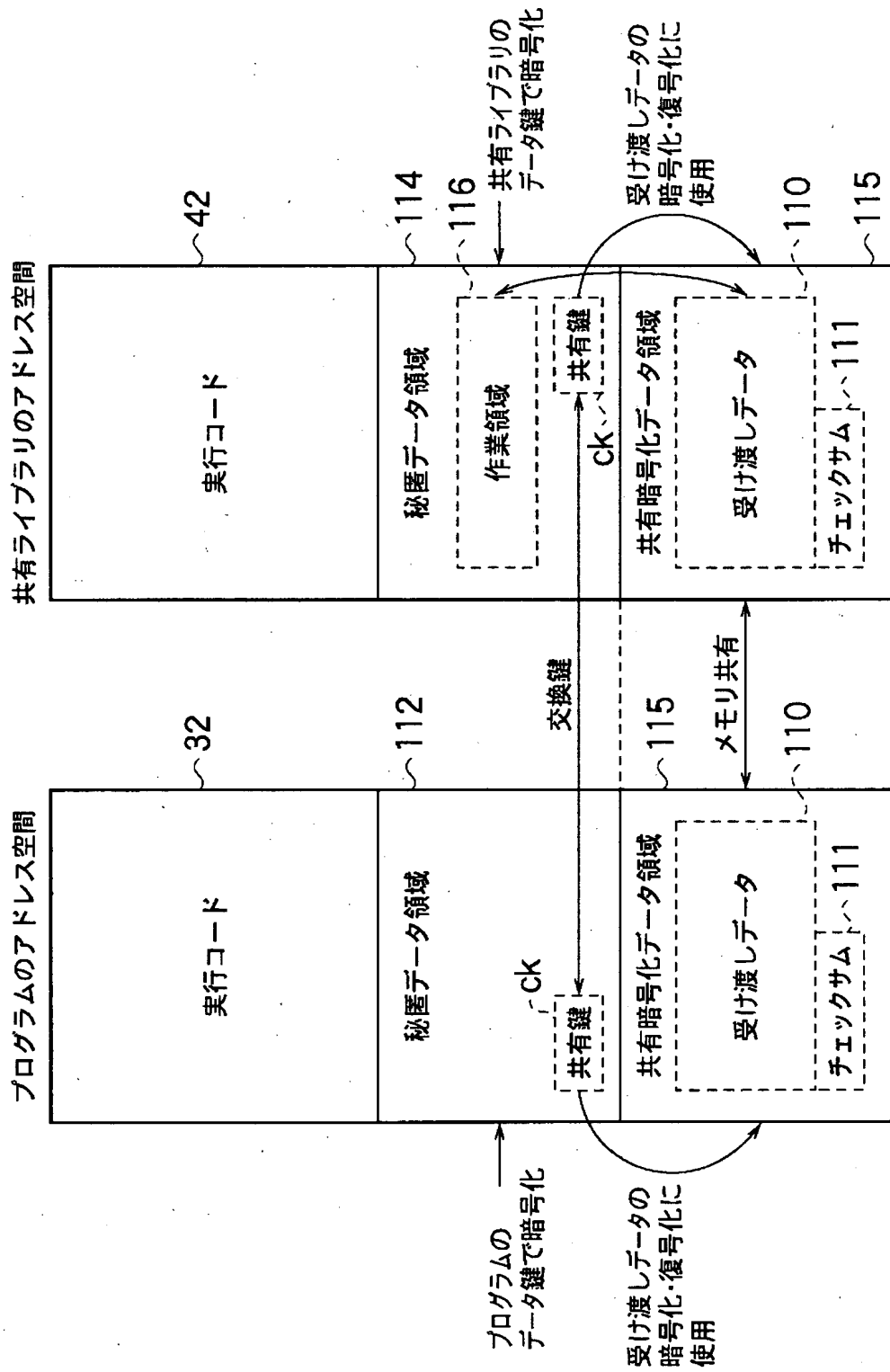
【図 9】



【図 1 0】



【図 11】



【書類名】 要約書

【要約】

【課題】 暗号化されたコードを復号化して実行する機能を有するマイクロプロセッサ上で動作し、自身の処理内容や処理結果を呼出元以外に漏洩しない共有ライブラリを実現する。

【解決手段】 暗号化されたコードを復号化して実行する機能を有し、各タスクに異なる識別子を与えて暗号鍵を格納するテーブルを有するマイクロプロセッサ 1 上で呼出元から共有ライブラリ 4 を使用可能にする方法は、共有ライブラリ用 4 のタスクを作成し、このタスクにタスク識別子を割り当てる。共有ライブラリ 4 のヘッダ 4 1 から命令鍵 k 4 1 を取得し、この命令鍵 k 4 1 を前記マイクロプロセッサ 1 内の前記共有ライブラリに割り当てられた識別子に対応する鍵値テーブル 1 3 に格納する。次に、共有ライブラリ 4 内のローダー 4 2 a を実行して初期化を行うステップと、共有ライブラリ 4 内のエントリポイント 4 2 b を経由して呼出元へ制御を返すステップとからなる。

【選択図】 図 9

特2002-192422

出 願 人 履 歴 情 報

識別番号

[000003078]

1. 変更年月日	2001年 7月 2日
[変更理由]	住所変更
住 所	東京都港区芝浦一丁目1番1号
氏 名	株式会社東芝